EnSight's External Command Driver

Computational Engineering International, Inc.
2166 N. Salem Street, Suite 101, Apex, NC 27523
USA • 919-363-0883 • 919-363-0833 FAX
http://www.ceintl.com or http://www.ensight.com

EN-CD Revision History

EN-CD:7.6-1	October 2002
EN-CD:7.6-2	April 2003

CEI's World Wide Web addresses:
http://www.ceintl.com
or
http://www.ensight.com

Restricted Rights Legend

EnSight Command Driver

This document provides information about a communication mechanism which can be used to drive EnSight from an external program using EnSight's command language. The logical steps involved in this process are:

1. **Compile your external program with the enscmddriver_comm library.**
2. **Start EnSight and have it listen for the connection from the external program**.
3. **Start the external program and issue the connect command within that program.**
4. **Send commands to EnSight using the enscmddriver_sendmesg routine**.
5. **Shutdown Ensight.**

More detail will now be provided for each of these steps.

Step 1:

Compile your external program with the enscmddriver.a library. This library is provided in the EnSight distribution, under the client_utilities/cmddriver directory. Directions for compiling are contained in the README file contained in that directory. Also provided therein is a sample external program (entitled enscmddriver.c) which is used to show how to compile, as well as for examples of how to utilize the following routines within your external driver:

enscmddriver_connect	To establish the connection with EnSight
enscmddriver_sendmesg	To send command language to EnSight
enscmddriver_query	To query information from EnSight (limited)
enscmddriver_disconnect	To disconnect and leave EnSight running (not commonly used)

Step 2:

Start EnSight and have it listen for the connection from the external program. Normally this will be done from your external program and will thus use batch mode to start EnSight.

In batch mode:

```
ensight7 -X -batch -externalcmds
```

While not the norm, it is possible to have EnSight start listening for the connection from an interactive session.

Interactively (from the command dialog in EnSight):

```
test: acceptcmddriver
```

Ensight will listen on Port 1104 for the connection from the external program. If a different port is desired, you can use the command line option "`-externalcmdport #`" when starting EnSight. Replace the `#` with a legitimate (>1024) port number. Then be sure to use the specified socket in the `enscmddriver_connect` call within your external program.

Step 3:

Once EnSight is listening, start the external program and issue the connect command within that program.

For the provided `enscmddriver` sample, this is done as follows:

```
enscmddriver HOSTNAME
```

Where, `HOSTNAME` is the name of the machine running EnSight. Note, the sample `enscmddriver` program calls the `enscmddriver_connect` routine to establish the connection.

Step 4:

Send commands to EnSight using the enscmddriver_sendmesg routine. The commands that you send to EnSight using this routine are the same commands that EnSight produces when users are manipulating a model with the EnSight Graphical User Interface. All of these commands are described in the Command Reference Manual within EnSight.

Note that the `enscmddriver_sendmesg` routine returns an `ok` or `ERROR` indicating its success or not.

It is possible to play entire command files that are accessible from the machine where the EnSight client is running. You can send a "play:" command to specify the name of the command file to use. Commands that are played using a file (play:) will execute faster than sending individual commands. The following is a command file (amiread.enc) that reads and colors the ami data set that is shipped with EnSight.

```
VERSION 7.52
data: binary_files_are big_endian
data: format case
data: path /usr/local/CEI/ensight74/data/ami
data: geometry ami.case
data: read
data_partbuild: begin
part: select_default
part: modify_begin
part: elt_representation not_loaded
part: modify_end
data_partbuild: data_type unstructured
data_partbuild: select_begin
 1
data_partbuild: select_end
data_partbuild: description
data_partbuild: create
part: select_default
part: modify_begin
part: elt_representation 3D_border_2D_full
part: modify_end
data_partbuild: data_type unstructured
data_partbuild: select_begin
 2
data_partbuild: select_end
data_partbuild: description
data_partbuild: create
data_partbuild: end
variables: activate pressure
part: select_all
part: modify_begin
part: colorby_palette pressure
part: modify_end
```

Your external program could send the command "play: amiread.enc" to Ensight using the `enscmddriver_sendmesg` routine. Ensight would play the command file, which would read in the model and color it by pressure, etc. It would then return and allow the external program to continue to issue other commands, such as would create images, produce VRML, create flipbook or keyframe animation sequences, etc.

Additionally, the `enscmddriver_query` routine can be used to obtain some limited information back from EnSight. The current possibilities for this option will be described in the query section below.

Step 5:

Shutdown Ensight. If you did the normal, and started ensight in batch mode - you close the communication and get Ensight to stop by sending an `exit` command with the `enscmddriver_sendmesg` routine.

If you happen to be running EnSight interactively, rather than the normal batch mode, and you desire to close the connection, but leave EnSight running - you can use the `enscmddriver_disconnect` routine.

Example

Assuming that you were able to successfully compile our sample external program, `enscmddriver`, and that your machine name was "speedy", you could do the following:

Start Ensight in batch mode (on your machine named "speedy"):

```
> ensight7 -X -batch -externalcmds &
```

Start the enscmddriver sample routine:

```
> enscmddriver speedy
```

Issue the following commands as prompted by the enscmddriver program:

```
What would you like to do?
play: amiread.enc
What would you like to do?
view: hidden_surface ON
What would you like to do?
savegeom: format vrml
What would you like to do?
savegeom: binary OFF
What would you like to do?
savegeom: save_geometric_entities /tmp/ami
What would you like to do?
exit
```

Which would read in the ami model using the amiread.enc command file, then turn shading on, then save a vrml file in /tmp. It would then close the communication and cause EnSight to exit.

You will of course be using your own external program, so the actual use of the enscmddriver_connect, and enscmddriver_sendmesg routines will be of interest to you. You can see them being used in the enscmddriver.c file. The routine arguments are described in detail in the Routine Descriptions section below.

Query Capability

The Ensight external command driver as first implemented with EnSight version 6.2.4, was purely a one-way interface. Namely, the external program could send command language to EnSight, but could not receive any type of information back (except for the error flag concerning success or failure of the command). Starting with EnSight 7.6, the capability to query EnSight for certain data has been added. While initially the scope of implemented queries is small, the implementation is general enough that future desirable queries should be easily added. Currently you can query for various transformation and viewport information.

The enscmddriver_query routine is driven by keywords. According to the keyword, the needed input parameters are defined, as well as the returned results.

Note: In the descriptions of the transformation matrices below, the components of a 4 x 4 matrix are:

```
| a11  a12  a13  a14 |
| a21  a22  a23  a24 |
| a31  a32  a33  a34 |
| a41  a42  a43  a44 |
```

```
The Composite Transformation matrix - A combination of the look_at/look_from transform and the
                           global transformation matrix.
Keyword:
  TRANSFORMATION_COMPOSITE_MATRIX
Input:
  param_array_cnt = 1
  param_array[0]  = Viewport number for the desired viewport (zero based)
Return Values:
  On Success -> (1)
    ret_float_cnt       = 16           (4 x 4 matrix)
    ret_float_array[0]  = a11    ret_float_array[8]  = a31
    ret_float_array[1]  = a12    ret_float_array[9]  = a32
    ret_float_array[2]  = a12    ret_float_array[10] = a32
    ret_float_array[3]  = a14    ret_float_array[11] = a34
    ret_float_array[4]  = a21    ret_float_array[12] = a41
    ret_float_array[5]  = a22    ret_float_array[13] = a42
    ret_float_array[6]  = a22    ret_float_array[14] = a42
    ret_float_array[7]  = a24    ret_float_array[15] = a44
  On Failure -> (-1)
    ret_error_buf contains the error message string
```

```
The Translate Transformation matrix

Keyword:
  TRANSFORMATION_TRANSLATE_MATRIX
Input:
  param_array_cnt = 1
  param_array[0]  = Viewport number for the desired viewport (zero based)
Return Values:
  On Success -> (1)
    ret_float_cnt       = 16           (4 x 4 matrix)
    ret_float_array[0]  = a11    ret_float_array[8]  = a31
    ret_float_array[1]  = a12    ret_float_array[9]  = a32
    ret_float_array[2]  = a12    ret_float_array[10] = a32
    ret_float_array[3]  = a14    ret_float_array[11] = a34
    ret_float_array[4]  = a21    ret_float_array[12] = a41
    ret_float_array[5]  = a22    ret_float_array[13] = a42
    ret_float_array[6]  = a22    ret_float_array[14] = a42
    ret_float_array[7]  = a24    ret_float_array[15] = a44
  On Failure -> (-1)
    ret_error_buf contains the error message string
```

The Scale Transformation matrix

```
Keyword:
  TRANSFORMATION_SCALE_MATRIX
Input:
  param_array_cnt = 1
  param_array[0]  = Viewport number for the desired viewport (zero based)
Return Values:
  On Success -> (1)
    ret_float_cnt       = 16           (4 x 4 matrix)
    ret_float_array[0]  = a11    ret_float_array[8]  = a31
    ret_float_array[1]  = a12    ret_float_array[9]  = a32
    ret_float_array[2]  = a12    ret_float_array[10] = a32
    ret_float_array[3]  = a14    ret_float_array[11] = a34
    ret_float_array[4]  = a21    ret_float_array[12] = a41
    ret_float_array[5]  = a22    ret_float_array[13] = a42
    ret_float_array[6]  = a22    ret_float_array[14] = a42
    ret_float_array[7]  = a24    ret_float_array[15] = a44
  On Failure -> (-1)
    ret_error_buf contains the error message string
```

The Rotate Transformation matrix

```
Keyword:
  TRANSFORMATION_SCALE_MATRIX
Input:
  param_array_cnt = 1
  param_array[0]  = Viewport number for the desired viewport (zero based)
Return Values:
  On Success -> (1)
    ret_float_cnt       = 16           (4 x 4 matrix)
    ret_float_array[0]  = a11    ret_float_array[8]  = a31
    ret_float_array[1]  = a12    ret_float_array[9]  = a32
    ret_float_array[2]  = a12    ret_float_array[10] = a32
    ret_float_array[3]  = a14    ret_float_array[11] = a34
    ret_float_array[4]  = a21    ret_float_array[12] = a41
    ret_float_array[5]  = a22    ret_float_array[13] = a42
    ret_float_array[6]  = a22    ret_float_array[14] = a42
    ret_float_array[7]  = a24    ret_float_array[15] = a44
  On Failure -> (-1)
    ret_error_buf contains the error message string
```

The Lookat Position

```
Keyword:
  TRANSFORMATION_LOOKAT_POSITION
Input:
  param_array_cnt = 1
  param_array[0]  = Viewport number for the desired viewport (zero based)
Return Values:
  On Success -> (1)
    ret_float_cnt = 3
    ret_float_array[0] = x coordinate of lookat point
    ret_float_array[1] = y coordinate of lookat point
    ret_float_array[2] = z coordinate of lookat point
  On Failure -> (-1)
    ret_error_buf contains the error message string
```

The Lookfrom Position

```
Keyword:
  TRANSFORMATION_LOOKFROM_POSITION
Input:
  param_array_cnt = 1
  param_array[0]  = Viewport number for the desired viewport (zero based)
Return Values:
  On Success -> (1)
    ret_float_cnt = 3
    ret_float_array[0] = x coordinate of lookfrom point
    ret_float_array[1] = y coordinate of lookfrom point
    ret_float_array[2] = z coordinate of lookfrom point
  On Failure -> (-1)
    ret_error_buf contains the error message string
```

The Center of Transformation

```
Keyword:
  TRANSFORMATION_CENTER_OF
Input:
  param_array_cnt = 1
  param_array[0]  = Viewport number for the desired viewport (zero based)
Return Values:
  On Success -> (1)
    ret_float_cnt = 3
    ret_float_array[0] = x coordinate of center of transformation
    ret_float_array[1] = y coordinate of center of transformation
    ret_float_array[2] = z coordinate of center of transformation
  On Failure -> (-1)
    ret_error_buf contains the error message string
```

The Location of bottom left of Viewport - returned both as screen and as normalized coords

```
Keyword:
  VIEWPORT_LOCATION
Input:
  param_array_cnt = 1
  param_array[0]  = Viewport number for the desired viewport (zero based)
Return Values:
  On Success -> (1)
    ret_float_cnt = 2
    ret_float_array[0] = normalized window x coordinate of bottom left of viewport (0. to 1.)
    ret_float_array[1] = normalized window y coordinate of bottom left of viewport (0. to 1.)
    ret_int_cnt = 3
    ret_int_array[0] = screen x coordinate of bottom left of viewport
    ret_int_array[1] = screen y coordinate of bottom left of viewport
  On Failure -> (-1)
    ret_error_buf contains the error message string
```

The Size of the Viewport, width and height - returned both as screen and as normalized values

```
Keyword:
  VIEWPORT_SIZE
Input:
  param_array_cnt = 1
  param_array[0]  = Viewport number for the desired viewport (zero based)
Return Values:
  On Success -> (1)
    ret_float_cnt = 2
    ret_float_array[0] = normalized window x size of viewport (0. to 1.)
    ret_float_array[1] = normalized window y size of viewport (0. to 1.)
    ret_int_cnt = 3
    ret_int_array[0] = screen x size of viewport
    ret_int_array[1] = screen y size of viewport
  On Failure -> (-1)
    ret_error_buf contains the error message string
```

The supplied sample external routine (`enscmddriver.c`) contains an example of the use of this routine.

Please see the Routine Description section for an explanation of the other arguments to the `enscmddriver_query` routine.

Routine Descriptions

```
/*****************************************************************************
 *  Starts up connection to the EnSight client to drive it via commands.
 *  Parameters:
 *     host_toconnectto       - Character buffer containing hostname
 *                              where EnSight is running.
 *     sockport               - Port number to use for socket( > 1024).
 *     print_error            - if (1) will print errors to stderr when
 *                              they occur.
 *  Return Values:
 *    On Success              - Socket file descriptor to communicate with
 *                              EnSight, if success.
 *    On Failure
 *      ENS_SOCKRANGE         - Port number out of range. Must be > 1024.
 *      ENS_CONNECT           - Connection to EnSight failed.  EnSight must
 *                              be ready for the external command connection.
 *      ENS_HANDSHAKE         - The call to receive the handshake string
 *                              from the EnSight client failed.
 *      ENS_HOSTTOOLONG       - The hostname specified is too large.
 ****************************************************************************/
int
enscmddriver_connect(char *host_toconnectto,
                     int   sockport,
                     int   print_error)
```

```
/*****************************************************************************
 *  This routine sends the EnSight client a command and waits for an ok (or ERROR).
 *  Parameters:
 *     comm_socket            - Socket to communicate on.
 *     cmd                    - command string being sent
 *     print_error            - if (1) will print errors to stderr when
 *                              they occur.
 *  Return Values:
 *      1 - upon success
 *     -1 - upon failure
 ****************************************************************************/
int
enscmddriver_sendmesg(int   comm_socket,
                      char *cmd,
                      int   print_error)
```

```
/*****************************************************************************
 *  This routine sends the EnSight client a query command and waits for the results.
 *  Parameters:
 *     comm_socket            - Socket to communicate on.
 *     query_keyword          - Query keyword
 *     param_array_cnt        - Count of parameters in array below.
 *     param_array            - Floating point array containing any parameters
 *                              for the query operation.  The count above helps
 *                              to clarify any changes that might be made to
 *                              a particular query in the future.  This will
 *                              help to allow forward/backward compatibility
 *                              and prevent users from always having to use
 *                              the latest library.
 *
 *     ***NOTE: the next 6 need to be passed in by address(ex. &ret_int_cnt)
 *              because return values will be placed in the ...cnt variables
 *              and space will be allocated for the others and return
 *              information will be placed in this space.
 *     ret_charstr_cnt        - Count of strings concatenated into string return
 *     ret_char_str           - String(s) returned from query and separated
 *                              by NULLs.  When the user finishes with the
 *                              information they must use free() to deallocate.
 *     ret_int_cnt            - Count of integers in return int array.
 *     ret_int_array          - Array of integer return values.   When the user
 *                              finishes with the information they must use
 *                              free() to deallocate.
 *     ret_float_cnt          - Count of floats in return float array.
 *     ret_float_array        - Array of float return values.   When the user
 *                              finishes with the information they must use
 *                              free() to deallocate.
 *
 *     ret_error_buf          - Buffer for error return string.  This buffer
 *                              should be preallocated to 500 characters by
 *                              the caller.  It will contain a NULL terminated
 *                              error string when the return value is -1.
 *
 *  Return Values:
 *    On Success              - (1)
 *    On Failure              - (-1)  (See error_buffer above)
 *
 ******************************************************************************/
int
enscmddriver_query(int     comm_socket,
                   char   *query_keyword,
                   int     param_array_cnt,
                   float  *param_array,
                   int    *ret_charstr_cnt,
                   char  **ret_char_str,
                   int    *ret_int_cnt,
                   int   **ret_int_array,
                   int    *ret_float_cnt,
                   float **ret_float_array,
                   char    ret_error_buf[500])



/*******************************************************************************
 * This routine cleans up the connection to EnSight.  This must
 * be done before you exit, especially if your application is dieing
 * because it received a signal.  If the socket is not closed properly
 * your port may become hung and you won't be able to use it until
 * it is cleared out by a reboot of your system or some other event.
 *
 *  Parameters:
 *     comm_socket            - Socket to communicate on.
 *
 *   Return Values:
 *      None
 ******************************************************************************/
void
enscmddriver_disconnect(int comm_socket)
```